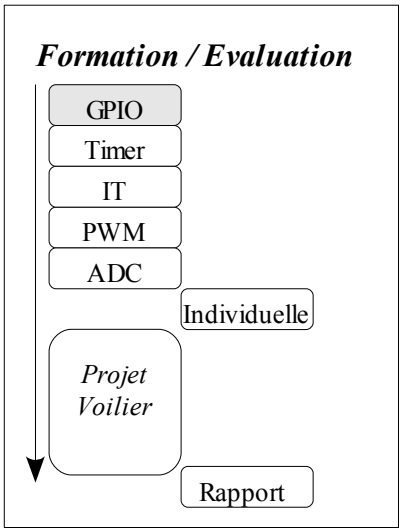
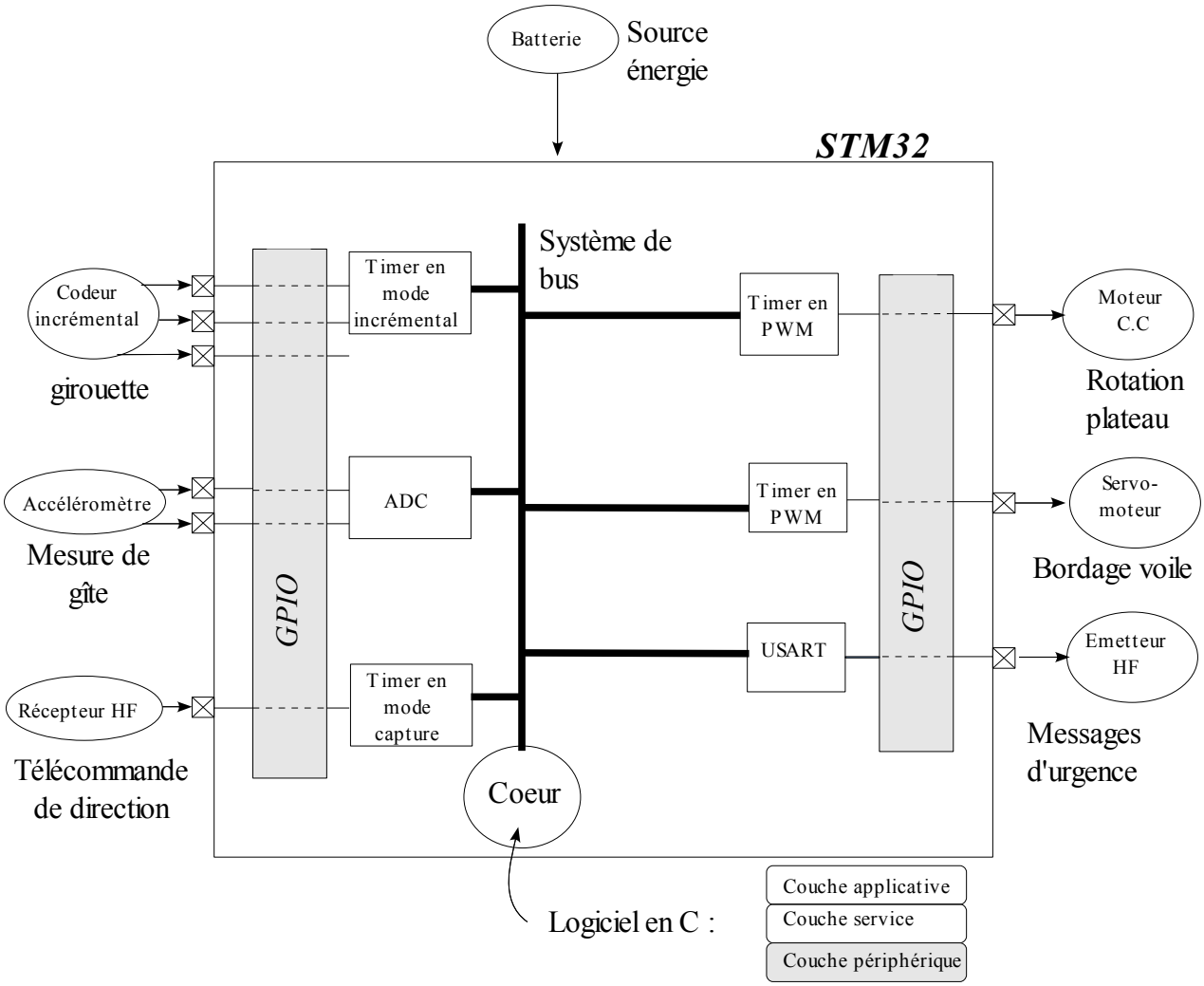


TP *port d'entrée sortie GPIO*

1- Le GPIO dans le système *Bordage de voile automatique d'un voilier*
2- Les ports d'E/S en général
3- Travail de l'étudiant – fonctionnement du GPIO du STM32
4- Travail de l'étudiant – KEIL

1 Le GPIO dans le système *Bordage de voile automatique d'un voilier*



2 Les ports d'E/S en général

Un port d'entrée/sortie communique avec l'extérieur du micro-contrôleur par le biais de plusieurs fils (broches), en général regroupé par paquet de 8 ou 16. Il communique avec le processeur par sa seule et unique possibilité : les bus d'adresses et de données. Ceci est commun à TOUS les périphériques du micro-contrôleur.
Par voie de conséquence, tout périphérique possède en son sein un certain nombre de registres accessibles en lecture ou écriture par le coeur. Ceux-ci on pour rôle de ***configurer*** et d'***utiliser*** le périphérique.

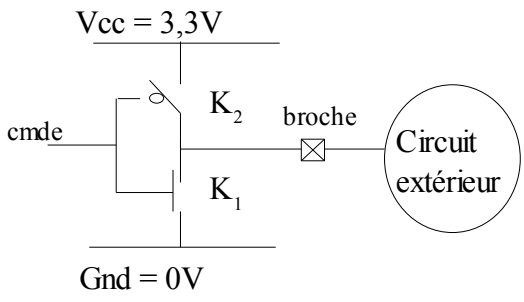
Un port d'entrée / sortie a donc pour rôle d'imposer (Output) ou de lire (Input) un niveau de tension (associé aux niveaux logique '0' ou '1') sur l'ensemble de ses broches. Selon le microcontrôleur, le niveau logique '1' peut être 5V, 3V3 ou encore 1V8.

Le port d'E/S possède donc au minimum deux registres de configuration (l'un qui spécifie pour chaque broche sa ***direction***, l'autre spécifiant la ***technologie*** utilisée) et un registre d'utilisation qui est à l'image logique des broches.

Parmi les technologies possibles on trouve :

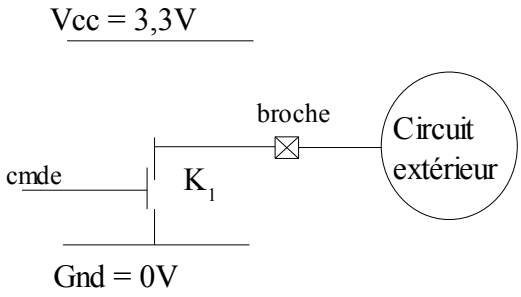
- **<u>En sortie :</u>**

Mode Push Pull :



La structure (technologie) possède deux interrupteurs (K_1, K_2, des transistors MOS complémentaires). K_1 et K_2 sont systématiquement inversés : La broche peut donc être portée au potentiel 0V ou 3,3V.

Mode Open Drain :

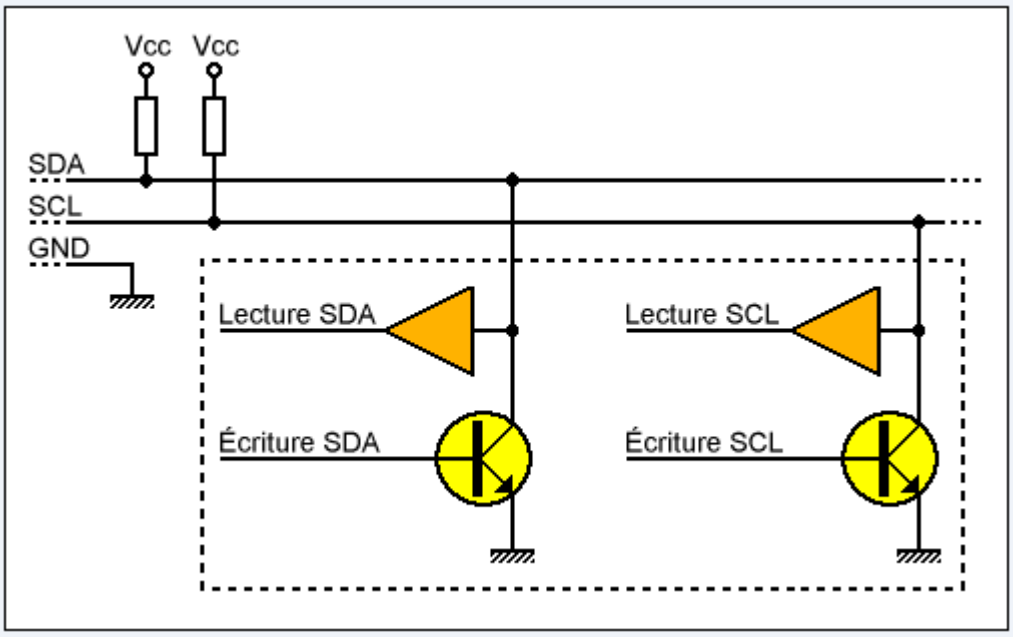


Ici, un seul intérrupteur est commandé, l'autre est maintenu bloqué (on ne le fait pas apparaître): la broche ne peut être portée <u>par le port</u> qu'à une tension de 0V.
Remarquons que si K_1 est ouvert, la broche est “en l'air”. Ce sera donc au <u>circuit extérieur</u> de fixer le potentiel de la broche dans cet état précis.

Exemple d'utilité du mode « Open Drain »

Le Bus I2C est un bus qui permet de mettre en communication plusieurs circuits numériques (micro-contrôleur, EEPROM, …). Deux fils sont utilisés *SCL* (horloge), *SDA* (donnée) sans oublier la masse. Il s'agit donc d'un bus série synchrone.

Voici l'exemple typique d'une structure à bus I2C:

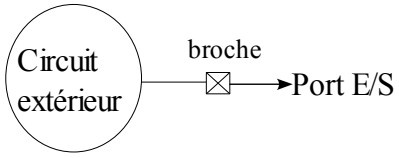


Nous voyons ici un seul circuit connecté au bus, il faut imaginer que n circuits sont susceptibles d'y être reliés de la même manière. Chaque circuit est composé de deux structures de lecture / écriture. En clair, chaque circuit peut prendre la main sur les lignes *SCL* et *SDA* (lecture ou écriture). Plusieurs circuits sont donc susceptibles d'émettre en même temps, c'est à dire de chercher à écrire sur la ligne *SDA* par exemple en même temps. Un conflit a donc lieu aux conséquences potentiellement néfastes sur le plan matériel (destruction) et logiciel (brouillage des messages) . Dans ce dernier cas, un arbitrage de bus permet de régler le problème.
Sur le plan matériel, il n'y a en réalité aucun problème non plus. En effet, les transistors (jaune) sont assimilables à des interrupteurs. On reconnaît donc une structure ***Open Drain***. Ainsi, chaque circuit peut imposer un '0' logique, mais jamais un '1'. C'est le circuit extérieur (la résistance de tirage) qui remplit ce rôle.
Si deux circuits imposent en même temps un '0', ou un '1', il n'y a pas de problèmes. Si un circuit impose un '0' et l'autre un '1', c'est celui qui impose le '0' qui « gagne », et le circuit qui proposait '1' n'est donc pas compris, mais il ne subit strictement aucun dégât. Si la technologie avait été *push-pull*, alors danc ce cas, un court-circuit se serait produit avec pour conséquence un courant trop fort débité par le circuit qui proposait '1' (ainsi qu'à celui qui proposait '0', c'est le plus fragile qui fait fusible...définitif).
Le montage *Open Drain*, d'un microcontrôleur permet donc à celui-ci de se relier à un bus I2C par exemple.

Résumons :
En mode **Push-Pull**, c'est le **port d'E/S qui impose** le niveau logique d'une broche, **que ce soit un niveau '1' ou '0'**. Il est le maître, le circuit extérieur est l'esclave, récepteur.
En mode **Open Drain** (*Drain* est le nom de la broche du transistor MOS reliée à la broche : *Drain laissé ouvert*), le port d'E/S ne peut **imposer que le niveau logique '0'**. Le niveau '1' est fixé par le circuit extérieur. Le port d'E/S n'est donc pas le seul maître du potentiel sur la broche. Le circuit extérieur l'est aussi.

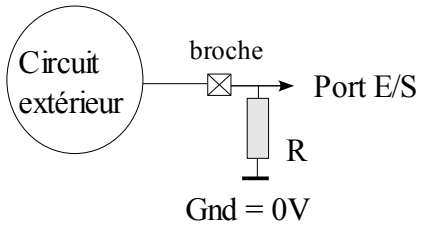
- **<u>En entrée :</u>**

***Mode entrée flottante (floating input)**:*



La broche, côté du port E/S, est laissée libre, flottante. Ainsi, c'est le circuit extérieur qui est totalement maître du potentiel de la broche.
Cela veut aussi dire, que si le circuit extérieur est déconnecté, le broche possède un potentiel inconnu (à proscrire car favorise le captage de parasites.

***Mode entrée tirée au niveau bas (Pull Down input)**:*



La broche, côté du port E/S, est reliée au 0V par l'intermédiaire d'une résistance (dite de rappel). L'avantage, c'est que si le circuit extérieur est déconnecté, le potentiel de la broche se retrouve à 0V grâce à la résistance de rappel.
Cela veut aussi dire, que le circuit extérieur, pour imposer un potentiel, doit avoir une résistance de sortie faible devant R, sinon, la tension chute. Elle est faussée.

***Mode entrée tirée au niveau haut (Pull up input)**:*
Même principe que précédemment, mais la résistance est reliée au Vcc (5V, 3V3, ou 1V8)

3 Travail de l'étudiant – fonctionnement du GPIO du STM32

Afin de réaliser les programmes logiciel de couche basse (périphérique), le programmeur doit avoir une connaissance solide du microcontrôleur cible. Ainsi, la première tâche est de chercher les informations nécessaires dans une documentation constructeur. Cette étape, longue, sera tout de même facilitée par les connaissances apportées au chapitre précédant.

3.1 La documentation à disposition

- ***"PM0056 Programming manual"***, document ST qui décrit le coeur *Cortex M3*, et en particulier les instruction assembleur mais aussi les périphériques natifs du coeur que sont le gestionnaire d'interruption, NVIC, et le timer système, SYSTICK. (*STM32_PM0056.pdf*)
- ***"RM0008 Reference manual"***, document ST qui décrit les périphériques de la famille des STM32F103xx, qui donne toutes les informations utiles au sujet des périphériques de la puce.
- ***"STM32F103x6, STM32F103x8, STM32F103xB"*** : datasheet du microcontrôleur qui traite de tout ce qui est limitation de vitesse, contraintes électrique, boîtier...

3.2 Méthodologie générale

1. **S'approprier le périphérique** en identifiant, dans le chapitre qui traite du périphérique étudié, sa structure, son fonctionnement, tout en s'appuyant sur que l'on sait déjà de ce type de circuit. Pour cela, très souvent on trouve un schéma fonctionnel qui aide beaucoup.

2. **Identifier les registres** qui sont nécessaires au fonctionnement du pérphérique, identifier les champs de bit dans chacun des registres qui ont une fonction de configuration intéressante pour l'application donnée.

3. **Noter en commentaire** dans les fichiers .c ces informations avant même de commencer à écrire le module de la couche driver.

3.3 Questionnaire guide

Vous devrez construire vous même ce questionnaire au fil des TP.

Nous allons travailler sur le microcontrôleur dont la référence est ***STM32F103RB***, en boîtier ***LQFP64***.

1- Donner le nombre de port GPIO de la puce, et pour chacun d'entre eux donner le nombre de broches associées

2- Trouver le nom des registres associés aux fonctionnalités suivantes :

	Fonctionnalités	***Nom du registre et bits***
Registres et bits de configuration	Permet de fixer la direction des broches du port	
	Permet du fixer la technologie de chaque broche	
Registres d'utilisation	Contient l'image du port en lecture	
	Contient l'image du port en écriture	

3- Expliquer à quoi sert le registre ***GPIOx.BSRR***

4- Décliner toutes les possibilités technologiques des broches du GPIO. Sur quel champ de registre doit-on agir pour faire le choix ?

5- Qu'appelle t-on ***Alternate function*** ?

4- Travail de l'étudiant – KEIL

Vous allez concevoir l'équivalent du “hello world” pour les périphériques, le “blinky, qui consiste à faire cligonter des leds. Vous utiliserez pour cela les masques binaires et la définition des registres.

Le projet sur lequel vous allez travailler est typiquement un exemple de fichier mélangeant les couches “pilotes” et “application”. Nous vous demandons dans la suite d'écrire un pilote des ports GPIO générique et de l'utiliser pour le blinky.

4.1 Structure des répertoires et démarrage

⌦ Chargez l'archive **paquet_etudiant.zip** depuis Moodle et dépaquetez la sur votre compte.

La structure de répertoire de l'archive est :

/tp_periph
- */sys*
- */pilotes*
  - */gpio*
- */projets*
  - */tp_gpio*
    - */src*
    - */release*

Le répertoire s*ys* contient les fichiers ***.h*** nécessaires à la définition des registres ainsi que le *fichier de démarrage* écrit en assembleur.
Le répertoire *pilotes* contiendra les modules pour configurer et piloter les périphériques.
Le projet microvision **TP_GPIO.uvproj** se trouve dans le répertoire *./projets/tp_gpio*. Les fichiers sources du projet sont dans *./projets/tp_gpio/src* et les fichiers résultants de la compilations sont placés dans *./projets/tp_gpio/release.*

⌦ Ouvrir le projet microvision **TP_GPIO.uvproj**

4.2 Compilation, test et exécution

⌦ Compiler le projet.
⌦ Exécuter le en simulation. Ouvrir la fenêtre des périphériques et regardez ce qui se passe en plaçant des points d'arrêt.
⌦ Passer en réel et constatez ce qui se passe.

4.2 Premier contact

⌦ Lire le code et comprendre.

⌦ Faire les TODO présents dans le code.

4.3 A vous de jouer

⌦ Ajouter la configuration en *input floating* du port **PA.0** qui est câblé sur le bouton **WKUP**.

⌦ Modifier l'application pour allumer la diode **PB.10** quand le bouton est maintenu appuyé tout en

conservant le clignotement de la diode **PB.9**.

4.4 Apprendre les bonnes manières

⌦ Ecrire une fonction qui permet de configurer une broche d'un port en output push-pull.
Le prototype de la fonction sera :

```
char Port_IO_Init_Output( GPIO_TypeDef * Port, u8 Broche)
```

⌦ Ecrire une fonction qui permet de configurer une broche d'un port en input floating.
Le prototype de la fonction sera :

```
char Port_IO_Init_Input( GPIO_TypeDef * Port, u8  Broche)
```

⌦ Ecrire une fonction qui permet de mettre à 1 une broche d'un port
Le prototype de la fonction sera :

```
void Port_IO_Set(GPIO_TypeDef * Port, u8 Broche)
```

⌦ Ecrire une fonction qui permet de mettre à 0 une broche d'un port
Le prototype de la fonction sera :

```
void Port_IO_Reset(GPIO_TypeDef * Port, u8 Broche)
```

⌦ Ecrire une fonction qui inverse l'état d'une broche
Le prototype de la fonction sera :

```
void Port_IO_Blink(GPIO_TypeDef * Port, u8 Broche)
```

⌦ Ecrire une fonction qui permet de lire l'état d'une broche d'un port
Le prototype de la fonction sera :

```
unsigned int Port_IO_Read(GPIO_TypeDef * Port, u8 Broche)
```

4,5 Faire un pilote

⌦ Créer un fichier **gpio.c** et un fichier d'en-tête **gpio.h** qui contient les fonctions de configuration et d'accès aux GPIO.

⌦ Pacez ces fichiers dans le répertoire *./pilotes/gpio* et modifier **Test_GPIO.c** pour ne faire appel qu'aux fonctions du pilote gpio.